CS170: Algorithms
Lecture in a minute.

Horn Formula:
- Implications of positive literals with ANDs on one side.
- Plus ORs of negatives.
- Negative clauses problem only with true literals.
- Greedy: only set true if have to.

Set Cover:
- Given subsets of some elements.
- Find: min number of sets that contains every element.
- Greedy: choose largest set w.r.t. remaining elements.
- \( O(\log n) \) approximate solution.
- Proof Idea: optimal of size \( k \Rightarrow \) Cover \( 1/k \) of the remaining elts.

Path Compression:
- \( O(m \log^* n) \) time for \( m \) finds.
- Some finds expensive but cheap on average.
- Idea: group ranks into \( \log^* n \) sets.
  - Small number of pointers across sets in any find.
  - Total movement inside sets \( O(n) \).
  - Idea: from not more than \( 2^k \) nodes of rank \( k \).
Taint Analysis

\[ a = \text{http.read_response}(); \]
\[ b = a + c; \]
\[ d = \text{sql.command}(b); \]

\(a\) is input from web.
“\(a\) is tainted.”
“if \(a\) is tainted \(b\) is tainted.”
“\(b\) should not be tainted.”

Logic representation:
\(A\) - “\(a\) is tainted”
\(B\) - “\(b\) is tainted”

\[ \implies A, A \implies B, \overline{B}. \]

Satisfiable?
Not in this case.
Horn SAT

Implications:

And of positive literals imply one positive literal.

\(x \land y \rightarrow z\)

Negative clauses:

Or of negative literals.

\(\bar{u} \lor \bar{v}\)

Taint Example:

\(\Longrightarrow A, A \Longrightarrow B, \bar{B}.\)

Is this satisfiable?

True is the problem.

If every literal is False:

All \(\land\) implication statements are good.

All \(\lor\) statements are true.

except for implication: \(\Longrightarrow A.\)

This forces a true literal.
Horn Sat: another view.

\[ x_1 \land x_2 \implies x_4 \]
\[ x_3 \implies x_2 \]
\[ x_1 \implies x_3 \]
\[ x_5 \land x_1 \implies x_3 \]
\[ x_2 \land x_6 \implies x_5 \]
\[ \implies x_1 \]

Problem: Find consistent assignment with fewest “True” literals.

Greedy algorithm: Only set literals to True if you have to.

Example:
\[ x_1 \] must be True so \[ x_3 \] must be True
so \[ x_2 \] must be True so \[ x_4 \] must be True
Solution: \{\( x_1, x_2, x_3, x_4 \)\} are True

Could also set \[ x_5 \] to true, or both \[ x_5 \] and \[ x_6 \] to true...but don’t!

Same as horn sat!
Why same as HornSAT?

Horn SAT had negative clauses.
No negative clauses for above algorithm.
Algorithm: Set a variable true ..if you have to!
Property: any variable set to true must be true in any satisfying assignment.
By induction. First $k$ set to true... must be!
The $k+1$ set variable set to true
is set to true to satisfy a clause
so it must be true.
Horn has negative clauses.
Negative clauses only problem for true variables.
Any variable that is true must be true.
So if a negative clause is false, it must be.
Any SAT formula?

\[ x_1 \implies x_2 \lor x_3. \]

\( x_1 \) being true may mean nothing for \( x_3 \)?
don’t have to set it to true.
No known polynomial time algorithm.
...no polynomial time algorithm unless \( \text{NP} = \text{P} \) ...
“\( \text{P} = \text{NP} \)”?

“There is an efficient algorithm to \textbf{find} a solution for every problem whose solution can be efficiently \textbf{checked}.”

More later...in the course.
Set Cover.

Input:
Items: $B = \{1, \ldots, n\}$
Sets: $S_1, \ldots, S_m \subseteq B$

Find fewest sets that cover $B$ (so that union is $B$)

Items: City Blocks.
Sets: Possible cellphone tower location.
Each cell phone tower location covers some subset of blocks.

Items: Customers.
Sets: Walmart locations covers subset of customers.

Items: Job responsibilities (ruby, perl, python, web, unix, ...).
Sets: People with job capabilities.

Items: Factory needs (touch screens, chips, cheap labor).
Sets: Suppliers.
(Thousands of suppliers for GM!!)
Greedy Algorithm

Choose set $S_i$ that has largest number of elts.
Remove elements in $S_i$ from all sets (Rinse).
Repeat.

Number of sets is number of iterations.
How many iterations?

Property: Set cover of size $k$ (best solution)
$\implies$ a set contains $\frac{1}{k}$ of remaining elements.

Analysis:
$n_t$ elements remain at time $t$ (after using $t$ sets.)
In iteration $t$, cover $\frac{1}{k} n_t$ remaining elements.

\[
\begin{align*}
    n_{t+1} &\leq n_t - \frac{1}{k} n_t = (1 - \frac{1}{k}) n_t. \\
    n_t &\leq (1 - \frac{1}{k})^t n_0
\end{align*}
\]

When do we stop?
Bound iterations.

When do we stop?

When \( n_t < 1 \)?

Recall: \( n_t \leq (1 - \frac{1}{k})^t n_0 \)

For what \( t \) must this be true?

(A) \( t = \log n \)

(B) \( t = k \)

(C) \( t = k \ln n \).

(C).

Plug in \( t = k \ln n \) and \( n_t < 1 \).

In more detail...
Bound iterations (really)

$$n_t \leq (1 - \frac{1}{k})^t n_0$$

When must $n_t < 1$?

Of course you remember: $(1 - x) \leq e^{-x}$ Smile!

So, $n_t \leq (1 - \frac{1}{k})^t n < (e^{-\frac{1}{k}})^t n \leq (e^{-\frac{t}{k}})n$.

For $t = k \ln n$, $n_t < (e^{-\ln n})n = (\frac{1}{n})n = 1$.

No elements are uncovered at this time!

So $t \leq k \ln n$. Number of sets for greedy is at most $k \ln n$!

Within $\ln n$ of $k$, which is the best possible!
We did not find optimal solution!
Is there a better analysis?
No. Problem 5.33!

Idea: Two sets cover all the elements.
One set covers slightly more than half the remaining elements.
Give $\Omega(\log n)$ lower bound.

Is there a better algorithm?
“Probably” not!
Again, only if $P=NP$.
More later in the course.
Disjoint Set Data Structure

Maintain pointers: $\pi(x)$ for each $x$. Initially: $\text{rank}(x) = 0$.

**makeset(x)** $\pi(x) = x$.

**find(x)**
- if $\pi(x) == x$
  - return $x$
- else
  - find($\pi(x)$)

**union(x,y)**
- $r_x = \text{find}(x)$
- $r_y = \text{find}(y)$
- if $\text{rank}(r_x) < \text{rank}(r_y)$:
  - $\pi(r_x) = r_y$
- else:
  - $\pi(r_y) = r_x$
  - if $\text{rank}(r_x) == \text{rank}(r_y)$:
    - $\text{rank}(r_x) += 1$

Properties:
1. Parent has a strictly higher rank.
2. Rank doesn’t change for internal nodes.
3. $\text{rank}(x) = \text{rank}(y) = k$
   - (i) Each have $\geq 2^k$ vertices in sets
   - (ii) and the sets are disjoint.
Path Compression

\[
\text{find}(x) \\
\text{if } \pi(x) == x \\
\hspace{1em} \text{return } x \\
\text{else} \\
\hspace{1em} \text{find}(\pi(x))
\]

What happens if we \text{find}(x) again? \\
Chase again!

\[
\text{find}(x) \\
\text{if } \pi(x) == x \\
\hspace{1em} \text{return } x \\
\text{else} \\
\hspace{1em} \pi(x) = \text{find}(\pi(x)) \\
\hspace{1.5em} \text{return } \pi(x)
\]
Is this better..

..asymptotically?

Take a deep breath.

Fancy stuff..next!

Don’t worry.

...do try..you’ll get smarter!
Path Compression Analysis

Union is same. Only affects root nodes.

Rank properties still hold.
   rank to parent is higher and $\geq 2^k$ node in rank $k$

Every find is asymptotically faster?

No. Can make a find take $\Theta(\log n)$ time.

Do you see how?

(A) Yes

(B) No

union(a,c) union(b,c)
   ... union(c,d)
union subtree roots to build tree
find(a)
$\Theta(\log n)$ time for this find.
Amortized Analysis.

Show that \( m \) finds take \( O(m \log^* n) \) time.

\( O(\log^* n) \) time on average!

\( \log^* n \) is number of times one takes \( \log \) to get to 1.

\( \log^*(16) \)?

(A) 4

(B) 2

(C) 3

C. \( \log 16 = 4, \log 4 = 2, \log 2 = 1 \). 3 times.

Also \( 2^{2^2} = 16 \). height of powers of two!

\( \log 1,000,000 \) versus \( \log^* 1,000,000 \)?

20 versus 5.

\( \log 1,000,000^1,000,000 \) versus \( \log^* 1,000,000^1,000,000 \)?

20,000,000 versus 6.

Grows very slowly.
Amortized Analysis.

Show that $m$ finds take $O(m \log^* n)$ time in total.
$O(\log^* n)$ time on average!
Amortize cost = average over many operations.
Who else amortizes?
Bankers!
Hand out some money
..... use it to pay for each pointer change.
Only hand out $O(m \log^* n)$ dollars.
Handing out dollars.

Will hand out money to internal nodes.....to pay for them changing pointers in find.

Notice: When a node becomes an internal node. rank will no longer change!

Divide non-zero ranks into levels.

\{1\}, \{2, 3, 4\}, \{5, \ldots, 16\} \ldots \{k + 1, \ldots 2^k\} \ldots

How many groups of ranks?

(A) $\Theta(\log n)$

(B) $\Theta(\log^* n)$

B. Each group grows by powering two!

How many internal nodes ever get rank $r$?

(A) $O(n/2^r)$

(B) $\Theta(n)$

A. Each contained $\geq 2^r$ nodes when root. Separate nodes.
Handing out money!

Will hand out money to internal nodes
.....since they change pointers in find.

Notice: When a node becomes an internal node.
    rank will no longer change!

If in set of ranks \( \{k + 1, \ldots, 2^k\} \) give node \( 2^k \) dollars.

\( O(n/2^r) \) internal nodes of rank \( r \).

Total Doled out:
    In a group: \( 2^k(n/2^{k+1} + n/2^{k+2} \cdots + n/2^{2^k}) = O(n) \).
    \( O(\log^* n) \) groups. Total money: \( O(n \log^* n) \).
Bounding find cost.

Bound cost of find operation.

\[ x \quad \rightarrow \quad z \quad \rightarrow \quad r \]

\text{rank}(x) \text{ and } \text{rank}(r) \text{ in different groups.}

\text{rank}(z) \text{ and } \text{rank}(r) \text{ in same group.}

\( O(1) \) plus

\text{cost of changing pointers to point to higher ranked nodes}

\textbf{Per Operation part.}

\( O(\log^* n) \) pointers that point to node to a higher group.

Total per operation cost over \( m \) finds: \( O(m \log^* n) \).

\textbf{Amortized Part.}

Node uses its dollars to pay for changing a pointer within group.

Recall group: \( \{k+1, \ldots, 2^{k+1}\} \)

\text{Enough money?}

\begin{itemize}
  \item only \( 2^k \) ranks in group.
  \item each node in group has \( 2^k \) dollars. \quad \text{Enough money!}
\end{itemize}

Total money: \( O(n \log^* n) \). \quad \Rightarrow \quad \text{Total find cost: } O((m + n) \log^* n)! !! !!
Intuition:

Some operations may be expensive.
...but modify data structure so they won’t be in future.
  (Path Compression!)

Place credits in data structure to pay for these modifications.

Still..

  fancy business.
Ackerman’s function.

Can we do better?

\[ f(k) \text{ is } 2^{2^k} \text{ of height } k. \]
Grows fast.

\[ f^{-1}(n) \text{ grows slowly! For example } f^{-1}((10^6)^{10^6}) = 5. \]

Ackermans function grows even faster: computable but grows faster than any primitive recursive function.

There is MST algorithm that runs in \( O(m\alpha(m, n)) \) where \( \alpha(m, n) \) is inverse Ackerman’s function.
Lecture in a minute.

Horn Formula:
Implications of positive literals with ANDs on one side.
Plus ORs of negatives.
Negative clauses problem only with true literals.
Greedy: only set true if have to.

Set Cover:
Given subsets of some elements.
Find: min number of sets that contains every element.
Greedy: choose largest set w.r.t. remaining elements.
$O(\log n)$ approximate solution.
Proof Idea: optimal of size $k \implies$ Cover $1/k$ of the remaining elts.

Path Compression:
$O(m \log^* n)$ time for $m$ finds.
Some finds expensive but cheap on average.
Idea: group ranks into $\log^* n$ sets.
Small number of pointers across sets in any find.
Total movement inside sets $O(n)$.
Idea: from not more than $2^k$ nodes of rank $k$. 